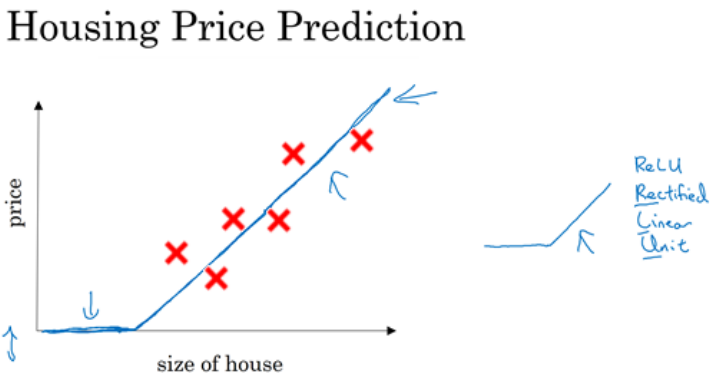
What is neural network?

It is a powerful learning algorithm inspired by how the brain works.

Example 1 – single neural network

Given data about the size of houses on the real estate market and you want to fit a function that will predict their price. It is a linear regression problem because the price as a function of size is a continuous output.

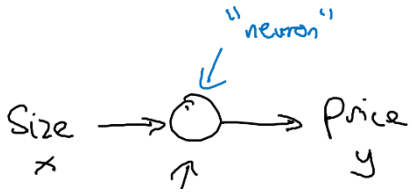
We know the prices can never be negative so we are creating a function called Rectified Linear Unit (ReLU) which starts at zero.



The input is the size of the house (x)

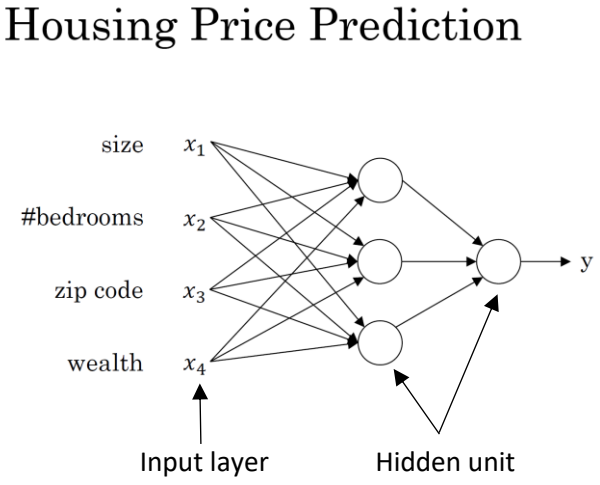
The output is the price (y)

The "neuron" implements the function ReLU (blue line)



Example 2 – Multiple neural network

The price of a house can be affected by other features such as size, number of bedrooms, zip code and wealth. The role of the neural network is to predicted the price and it will automatically generate the hidden units. We only need to give the inputs x and the output y.



Supervised learning for Neural Network

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

Here are some examples of supervised learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

standard NN → Real Estate, Online Advertising
CNN → Photo tagging
RNN → Speech recognition, Machine translation
hybrid → Autonomous driving

There are different types of neural network, for example Convolution Neural Network (CNN) used often for image application and Recurrent Neural Network (RNN) used for one-dimensional sequence data such as translating English to Chinses or a temporal component such as text transcript. As for the autonomous driving, it is a hybrid neural network architecture.

Structured vs unstructured data

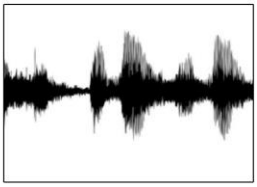
Structured data refers to things that has a defined meaning such as price, age whereas unstructured data refers to thing like pixel, raw audio, text.

Structured Data

Size	#bedrooms	...	Price (1000$s)
2104	3		400
1600	3		330
2400	3		369
⋮	⋮		⋮
3000	4		540

User Age	Ad Id	...	Click
41	93242		1
80	93287		0
18	87312		1
⋮	⋮		⋮
27	71244		1

Unstructured Data



Audio

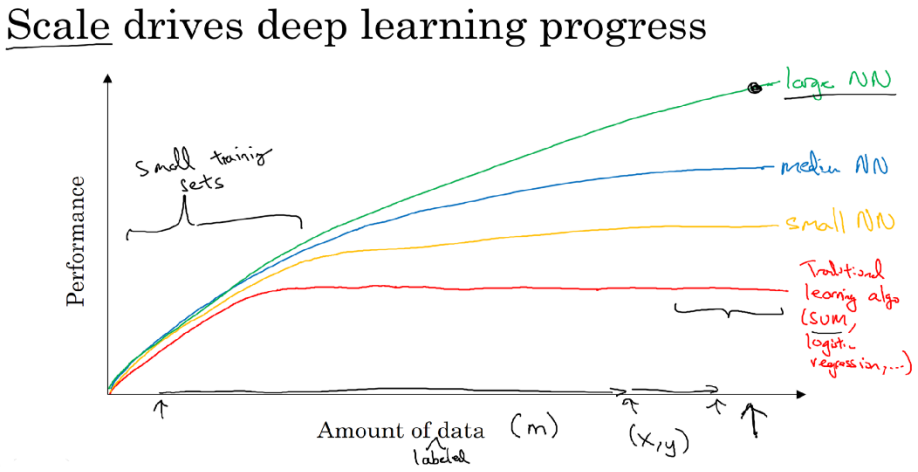
Image

Four scores and seven years ago...

Text

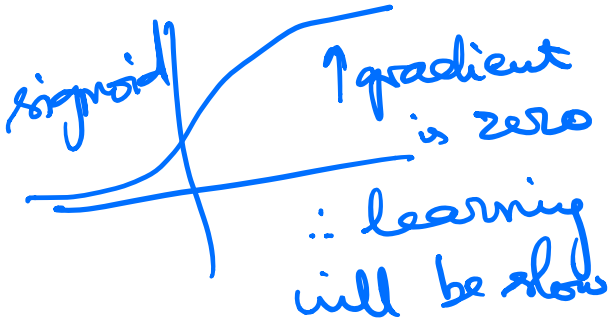
Why is deep learning taking off?

Deep learning is taking off due to a large amount of data available through the digitization of the society, faster computation and innovation in the development of neural network algorithm.

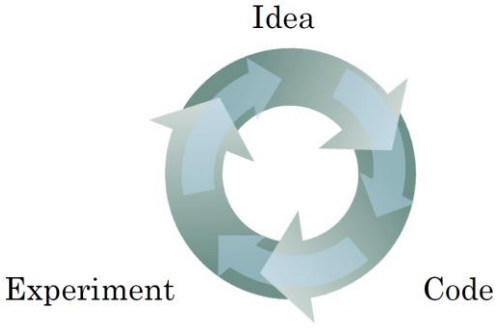


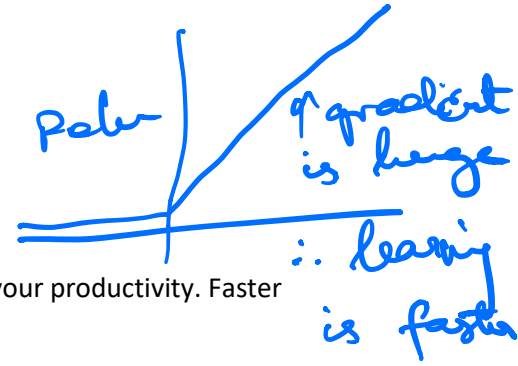
Two things have to be considered to get to the high level of performance:

1. Being able to train a big enough neural network
2. Huge amount of labeled data



The process of training a neural network is iterative.





It could take a good amount of time to train a neural network, which affects your productivity. Faster computation helps to iterate and improve new algorithm.

Standard notations for Deep Learning

This document has the purpose of discussing a new standard for deep learning mathematical notations.

1 Neural Networks Notations.

General comments:

· superscript (i) will denote the i^{th} training example while superscript [l] will denote the l^{th} layer

Sizes:

·m : number of examples in the dataset

·n_x : input size

·n_y : output size (or number of classes)

·$n_h^{[l]}$: number of hidden units of the l^{th} layer

In a for loop, it is possible to denote $n_x = n_h^{[0]}$ and $n_y = n_h^{[\text{number of layers } +1]}$.

·L : number of layers in the network.

Objects:

·$X \in \mathbb{R}^{n_x \times m}$ is the input matrix

·$x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th}example represented as a column vector

·$Y \in \mathbb{R}^{n_y \times m}$ is the label matrix

·$y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example

·$W^{[l]} \in \mathbb{R}^{\text{number of units in next layer} \times \text{number of units in the previous layer}}$ is the weight matrix,superscript [l] indicates the layer

·$b^{[l]} \in \mathbb{R}^{\text{number of units in next layer}}$ is the bias vector in the l^{th} layer

·$\hat{y} \in \mathbb{R}^{n_y}$ is the predicted output vector. It can also be denoted $a^{[L]}$ where L is the number of layers in the network.

Common forward propagation equation examples:

$a = g^{[l]}(W_x x^{(i)} + b_1) = g^{[l]}(z_1)$ where $g^{[l]}$ denotes the l^{th} layer activation function

$\hat{y}^{(i)} = softmax(W_h h + b_2)$

· General Activation Formula: $a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]})$

· $J(x, W, b, y)$ or $J(\hat{y}, y)$ denote the cost function.

Examples of cost function:

· $J_{CE}(\hat{y}, y) = -\sum_{i=0}^{m} y^{(i)} \log \hat{y}^{(i)}$

· $J_1(\hat{y}, y) = \sum_{i=0}^{m} \mid y^{(i)} - \hat{y}^{(i)} \mid$

2 Deep Learning representations

For representations:

· nodes represent inputs, activations or outputs

· edges represent weights or biases

Here are several examples of Standard deep learning representations

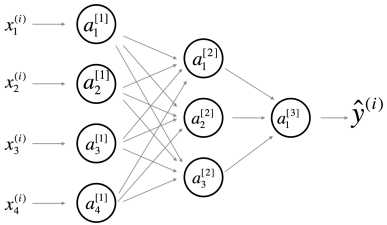


Figure 1: Comprehensive Network: representation commonly used for Neural Networks. For better aesthetic, we omitted the details on the parameters ($w_{ij}^{[l]}$ and $b_i^{[l]}$ etc...) that should appear on the edges

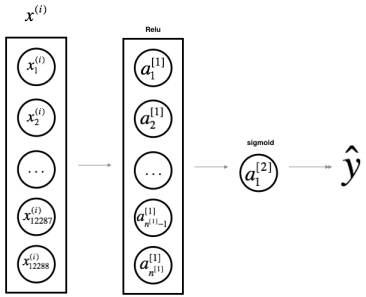


Figure 2: Simplified Network: a simpler representation of a two layer neural network, both are equivalent.

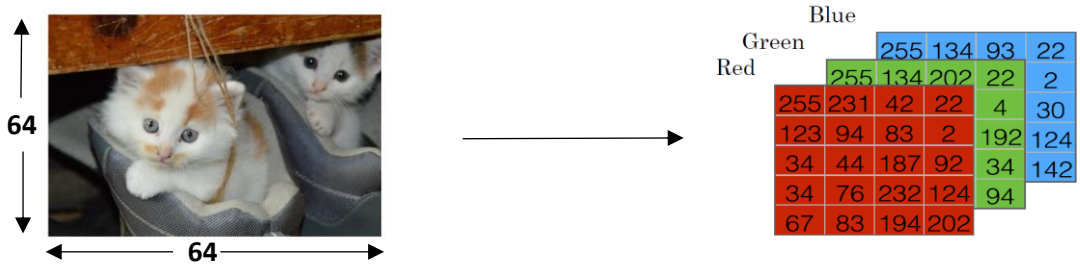
Binary Classification

In a binary classification problem, the result is a discrete value output.

For example - account hacked (1) or compromised (0)
- a tumor malign (1) or benign (0)

Example: Cat vs Non-Cat

The goal is to train a classifier that the input is an image represented by a feature vector, x, and predicts whether the corresponding label y is 1 or 0. In this case, whether this is a cat image (1) or a non-cat image (0).



An image is store in the computer in three separate matrices corresponding to the Red, Green, and Blue color channels of the image. The three matrices have the same size as the image, for example, the resolution of the cat image is 64 pixels X 64 pixels, the three matrices (RGB) are 64 X 64 each.

The value in a cell represents the pixel intensity which will be used to create a feature vector of n-dimension. In pattern recognition and machine learning, a feature vector represents an object, in this case, a cat or no cat.

To create a feature vector, x, the pixel intensity values will be "unroll" or "reshape" for each color. The dimension of the input feature vector x is $n_x = 64\ x\ 64\ x\ 3 = 12\ 288$.

$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \text{red} \\ \\ \\ \\ \text{green} \\ \\ \\ \\ \text{blue} \\ \\ \end{matrix}$$

Notation

(x, y) $\quad x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

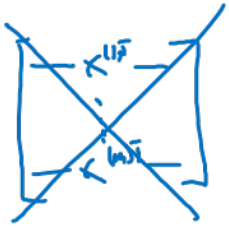
m training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$M = M_{train}$ $\qquad$ M_{test} = #test examples.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

n_x

m



$X \in \mathbb{R}^{n_x \times m}$ $\qquad$ $X.shape = (n_x, m)$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

$Y \in \mathbb{R}^{1 \times m}$

Y shape $= (1, m)$

Logistic Regression

Logistic regression is a learning algorithm used in a supervised learning problem when the output y are all either zero or one. The goal of logistic regression is to minimize the error between its predictions and training data.

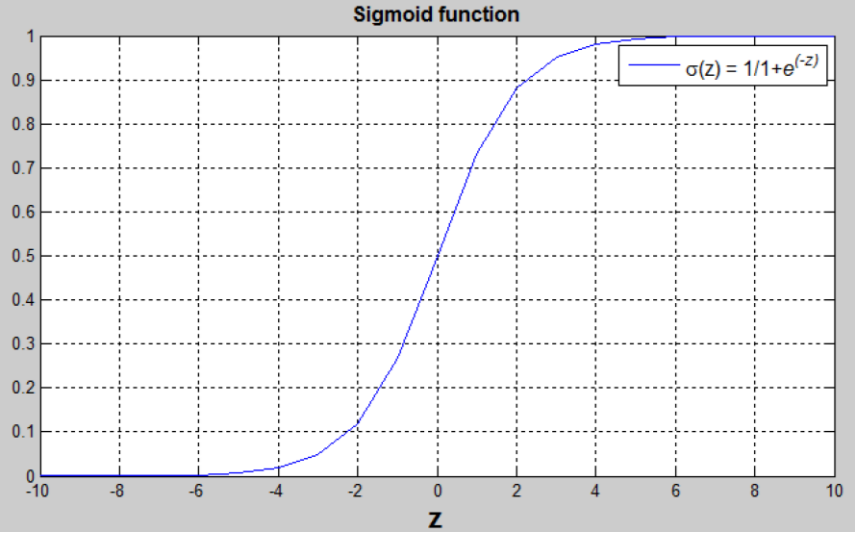
Example: Cat vs No - cat

Given an image represented by a feature vector x, the algorithm will evaluate the probability of a cat being in that image.

$$Given\ x\ ,\ \hat{y} = P(y = 1|x), \text{where } 0\ \leq \hat{y} \leq 1$$

The parameters used in Logistic regression are:

- The input features vector: $x\ \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The training label: $y \in 0,1$
- The weights: $w\ \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The threshold: $b\ \in \mathbb{R}$
- The output: $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function: s = $\sigma(w^T x + b)$ = $\sigma(z)$= $\frac{1}{1+ e^{-z}}$



$(w^T x + b)$ is a linear function $(ax + b)$, but since we are looking for a probability constraint between [0,1], the sigmoid function is used. The function is bounded between [0,1] as shown in the graph above.

Some observations from the graph:

- If z is a large positive number, then $\sigma(z)$ = 1
- If z is small or large negative number, then $\sigma(z)$ = 0
- If $z = 0$, then $\sigma(z)$ = 0.5

Logistic Regression: Cost Function

To train the parameters w and b, we need to define a cost function.

Recap:

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

$x^{(i)}$ the i-th training example

$Given \{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\},\ we\ want\ \hat{y}^{(i)} \approx y^{(i)}$

Loss (error) function:

The loss function measures the discrepancy between the prediction ($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$). In other words, the loss function computes the error for a single training example.

$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$

$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$

- If $y^{(i)} = 1$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$ where $\log(\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$ where $\log(1 - \hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 0

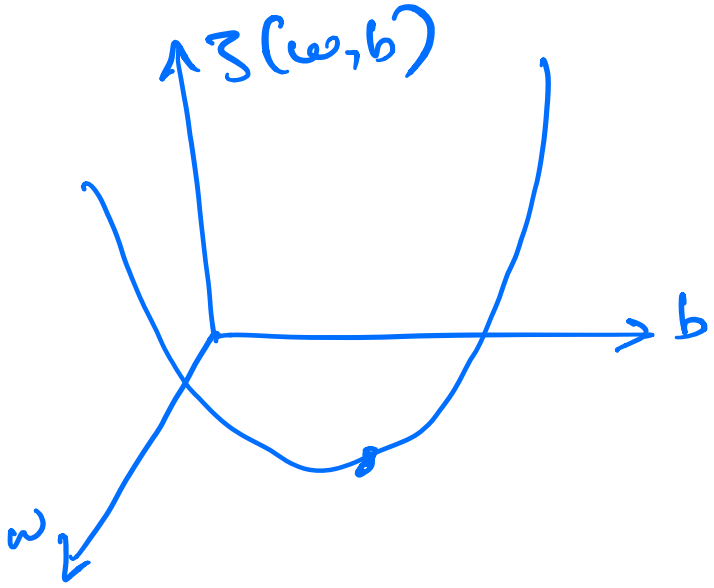
Cost function

The cost function is the average of the loss function of the entire training set. We are going to find the parameters w and b that minimize the overall cost function.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

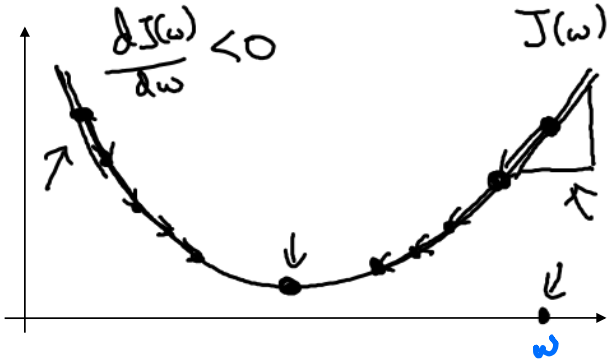
Gradient descent:

find w and b that minimize J(w,b)



It has only one optimal point unlike curves like

Gradient Descent



$\frac{dJ(w)}{dw} > 0$

Repeat {

$w := w - \alpha \frac{dJ(w)}{dw}$

learning rate

"dw"

}

$w := w - \alpha dw$

$\frac{dJ(w)}{dw} = ?$

$J(w,b)$

$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$ $\quad \frac{\partial J(w,b)}{\partial w} \rightarrow dw$

$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$ $\quad \frac{\partial J(w,b)}{\partial b} \rightarrow db$

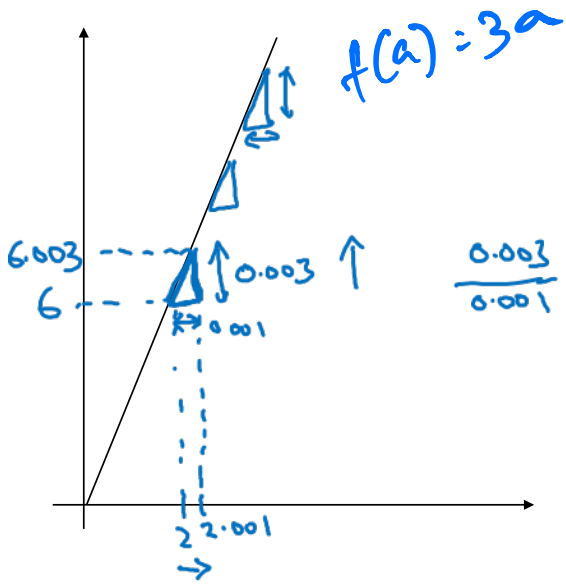
∂ "partial derivative"

∂J

α - how big a step in gradient descent

$\frac{\partial J(w,b)}{\partial w}$ → slope in w direction

Intuition about derivatives



$\frac{0.003}{0.001}$ $\frac{\text{height}}{\text{width}}$

$a = 2$ $\quad f(a) = 6$

$a = 2.001$ $\quad f(a) = 6.003$

slope (derivative) of $f(a)$ at $a = 2$ is 3

$a = 5$ $\quad f(a) = 15$

$a = 5.001$ $\quad f(a) = 15.003$

slope at $a = 5$ is also 3

$\frac{d f(a)}{da} = 3 = \frac{d}{da} f(a)$

$f(a)$ changes

0.001

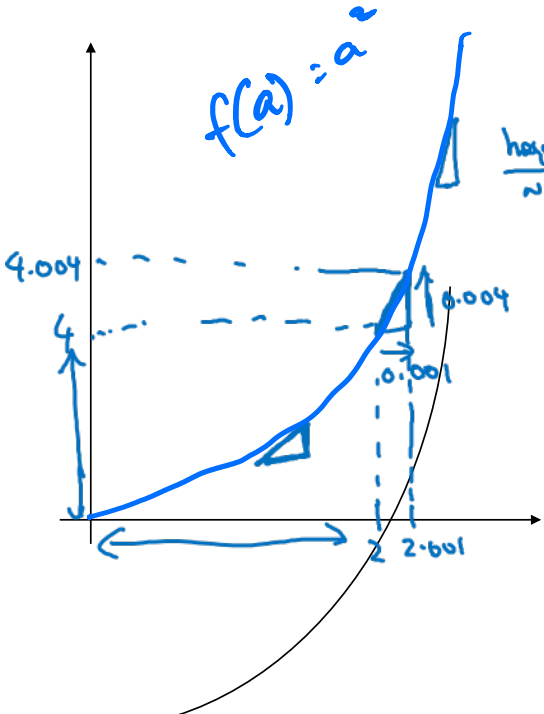
0.00000001

0.00000000001

more a

Andrew Ng

Intuition about derivatives



$\frac{d}{da} a^2 = 2a$

0.001

$(2a) \times 0.001$

0.001 ←

$0.00000\ldots01$ ←

$a = 2$ $\quad f(a) = 4$

$a = 2.001$ $\quad f(a) \approx 4.004$ ↓ $(4.004\boxed{004})$

slope (derivative) of $f(a)$ at $a = 2$ is 4.

$\boxed{\frac{d}{da} f(a) = 4}$ when $\boxed{a = 2}$.

$a = 5$ $\quad f(a) = 25$

$a = 5.001$ $\quad f(a) \approx 25.010$

$\boxed{\frac{d}{da} f(a) = 10}$ when $\boxed{a = 5}$

$\frac{d}{da} f(a) = \frac{d}{da} a^2 = \boxed{2a}$

More derivative examples

$f(a) = a^2$ $\quad \frac{d}{da} f(a) = 2a = 4$

$a = 2$ $\quad f(a) = 4$

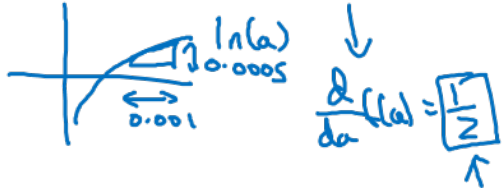
$a = 2.001$ $\quad f(a) \approx 4.004$

$f(a) = a^3$ $\quad \frac{d}{da} f(a) = 3a^2$ $\quad 3 * 2^2 = 12$

$a = 2$ $\quad f(a) = 8$

$a = 2.001$ $\quad f(a) \approx 8.012$

$f(a) = \log_e(a)$ $\quad \ln(a)$ $\quad \frac{d}{da} f(a) = \frac{1}{a}$



$a = 2$ $\quad f(a) \approx 0.69315$

$a = 2.001$ $\quad f(a) \approx 0.69365$

0.0005 $\quad 0.0005$

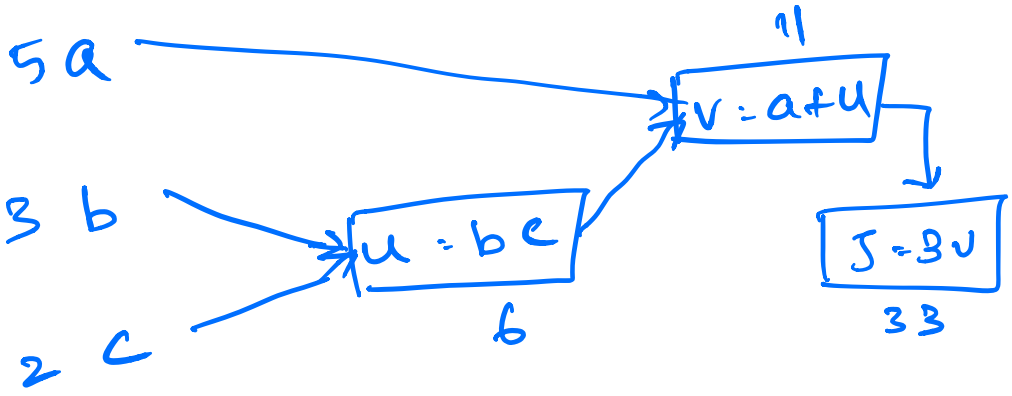
computation graph:

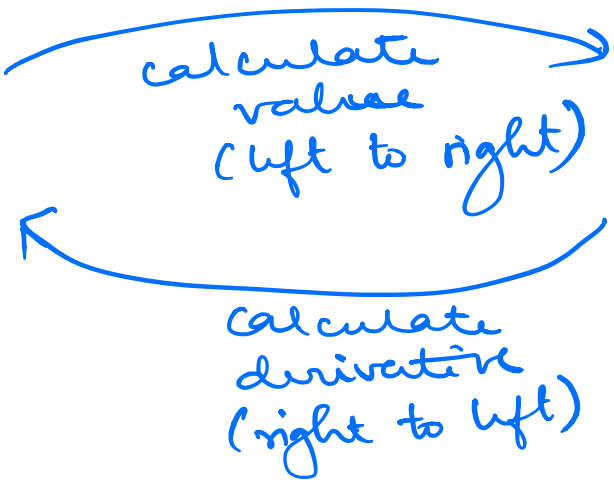
$$J(a,b,c) = 3(a+b*c)$$

$$u = bc$$

$$v = a+u$$

$$J = 3*v$$





$\frac{dJ}{dv}$ → how much will J value change if i modify v

dv → notation for $\frac{dJ}{dv}$

$J = 3v \rightarrow 33.003$

$v = 11 \rightarrow 11.001$

$\therefore \frac{dJ}{dv} = 3$

$\frac{dJ}{da}$ ← chain rule

$a = 5 \rightarrow 5.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$\frac{dv}{da}$ $\frac{dJ}{dv}$

$\frac{0.003}{0.001} = 3$

$\frac{0.001}{0.001} = 1$

$= 3$

$\frac{dJ}{da} = 3$

$\frac{dJ}{du}$

$\frac{dJ}{dv}$ $\frac{dv}{du}$

6 1

3 $= 3$

$u = 6 \rightarrow 6.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$\frac{dJ}{du} = 3$

$\frac{dJ}{db} = \underbrace{\frac{dJ}{du}}_{=3} \underbrace{\frac{du}{db}}_{=2}$

$= 6$

$b = 3.001$

$u = 6.002$, $c = 2$

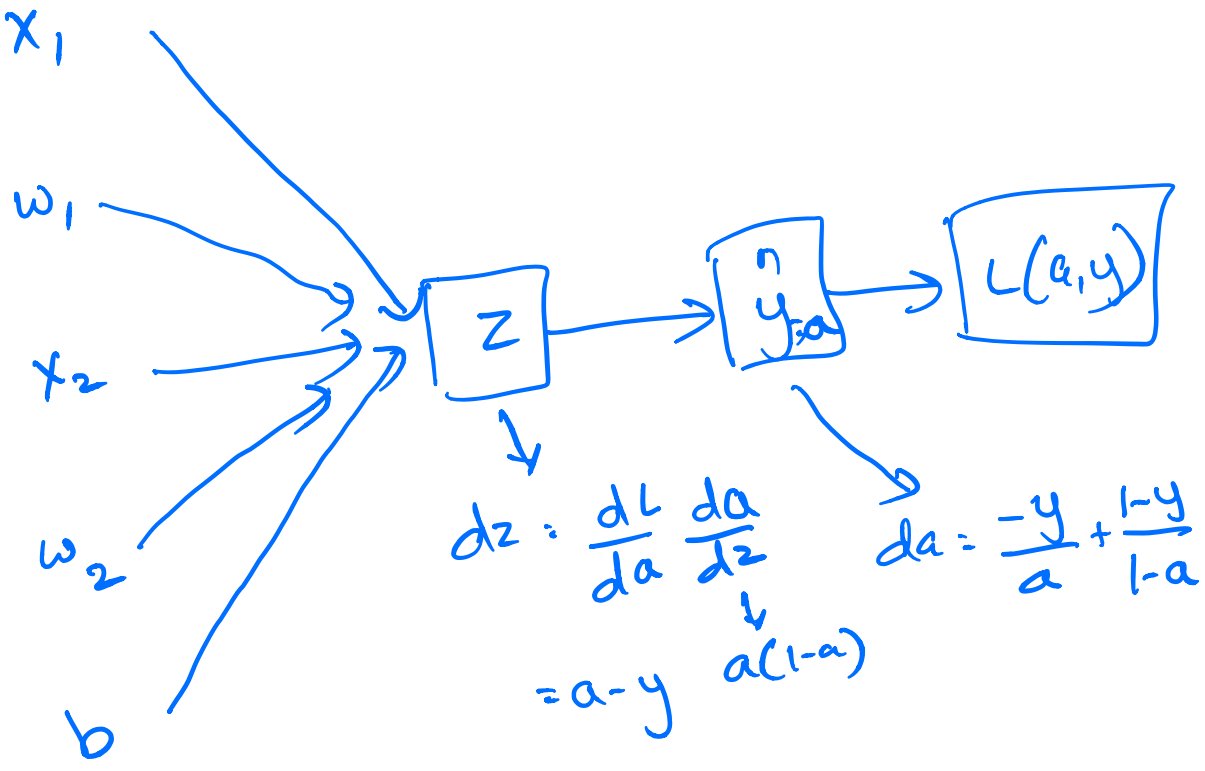
$\frac{dJ}{dc} = \underbrace{\frac{dJ}{du}}_{3} \underbrace{\frac{du}{dc}}_{3} = 9$

Logistic regression: modify w, b to reduce L

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$L(a, y) = -(y \log(a) + (1-y) \log(1-a))$$



$$\frac{dL}{dw_1} = x_1 \cdot dz \qquad dw_2 = x_2 dz$$

$$db = dz$$

$$w_1 = w_1 - \alpha dw_1$$
$$w_2 = w_2 - \alpha dw_2$$
$$b = b - \alpha db$$

} update eqn.

for m training examples: (one step)

$J = dw_1 = dw_2 = db = 0$

for $i = 1$ to m

calculate $z^{(i)}$, $a^{(i)}$

$J \mathrel{+}= -[y^i \log a^i + (1-y^i) \log(1-a^i)]$

$dz^{(i)} = a^i - y^i$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J /= m$, $dw_1 /= m$, $dw_2 /= m$, $db /= m$

then we update eqn

Derivations:

$\frac{dL}{da}$: $L = -[y \log a + (1-y) \log(1-a)]$

wrt a:

$\frac{dL}{da} = -y \times \frac{1}{a} - (1-y) \frac{1(-1)}{1-a}$

$= \frac{-y}{a} + \frac{1-y}{1-a}$

$= \frac{-y + ay + a - ay}{a(1-a)}$

$= \frac{a-y}{a(1-a)}$

$\frac{d}{dz} \tanh(z) = 1 - (\tanh z)^2$

$\frac{d}{dz}$ relu $= \begin{cases} 0, & z<0 \\ 1, & z \geq 0 \\ NA, & z=0 \end{cases}$

$\frac{d}{dz}$ Leaky relu $= \begin{cases} 0.01, & z<0 \\ 1, & z>0 \\ NA, & z=0 \end{cases}$

$\frac{da}{dz}$: $a = \sigma(z) = \frac{1}{1+e^{-z}}$

wrt z:

$\frac{da}{dz} = \frac{-1}{(1+e^{-z})^2} \frac{d}{dz}(1+e^{-z})$

$= \frac{-1}{(1+e^{-z})^2}(-e^{-z})$

$f = \frac{1}{x}$

$f' = \frac{-1}{x^2}$

$$= \frac{+e^{-z}}{(1+e^{-z})^2} \quad \text{—①}$$

$$\sigma(z) * (1-\sigma(z)) = \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= \frac{1}{1+e^{-z}} \left(\frac{1+e^{-z}-1}{1+e^{-z}}\right)$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} \quad \text{—②}$$

from ① and ② $\frac{da}{dz} = \sigma(z)(1-\sigma(z))$

$$= a(1-a)$$

$$\frac{dL}{dz} = \frac{dL}{da}\frac{da}{dz} = \frac{a-y}{a(1-a)} * a(1-a)$$

$$= a - y$$

$$\frac{dz}{dw_i} : \quad z = w_1 x_1 + w_2 x_2 + \cdots + b$$

wrt w_i wrt b

$$\frac{dz}{dw_i} = x_i \qquad \frac{dz}{db} = 1$$

$$\therefore \frac{dL}{dw_i} = \frac{dL}{dz} \frac{dz}{dw_i}$$

$\frac{dL}{dz}$ ↑ dz

$$= x_i \, dz$$

$$\text{Similarly } \frac{dz}{db} = dz$$

Calculation: (vector) (one iteration of gradient descent)

$$z = w^T X + b$$

$$A = \sigma(z)$$

$$dz = A - Y$$

$$dw = \frac{1}{m} X \, dz^T$$

$$db = \frac{1}{m} \text{sum}(z)$$

$X = n_x \times m$

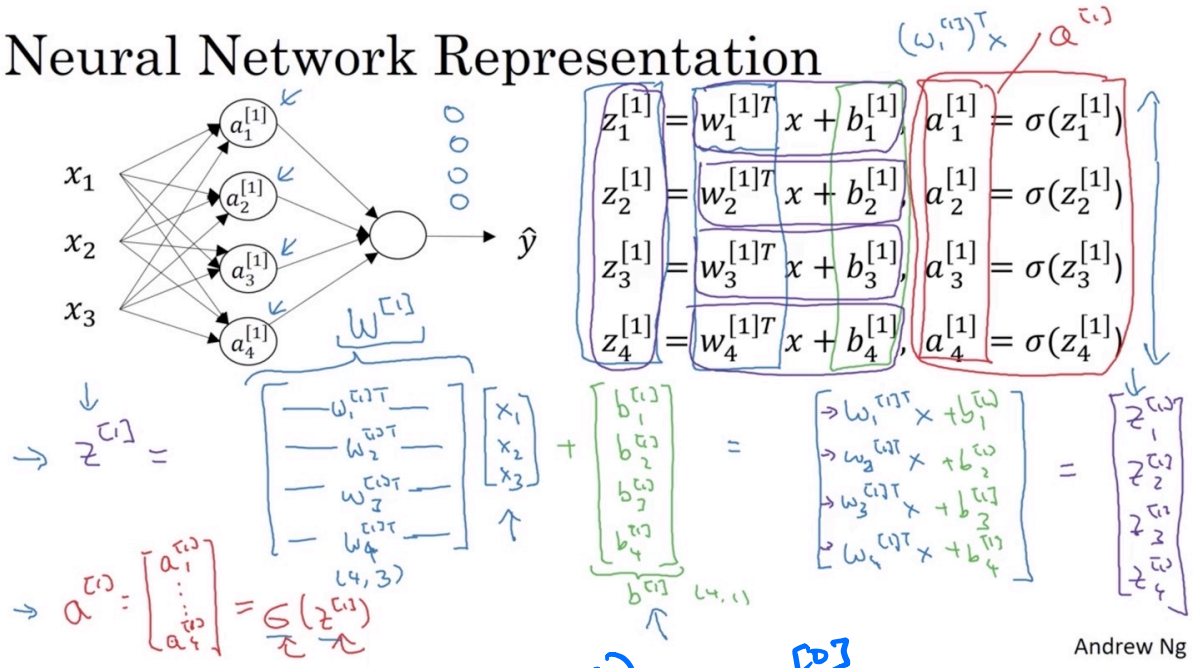
$w : w - \alpha dw$

$b : b - \alpha db$

Neural networks:

$[l]$ ← layer l

a_i ← i^{th} node in that layer



Given x:

$$z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} x_{(3,1)} + b^{[1]}_{(4,1)}$$

$$a^{[1]}_{(4,1)} = \sigma(z^{[1]})$$

$$z^{[2]}_{(1,1)} = W^{[2]}_{(1,4)} a^{[1]}_{(4,1)} + b^{[2]}_{(1,1)}$$

$$a^{[2]}_{(1,1)} = \sigma(z^{[2]})$$

for m training examples:

$X \rightarrow (n_x, m)$

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

$$A^{[1]} = \begin{bmatrix} a^{[1][1]}_{1} & a^{[1][2]}_{1} & \cdots & a^{[1][m]}_{1} \\ | & | & & | \end{bmatrix}$$

↓ diff hidden units

→ diff training sample

$$\tanh = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

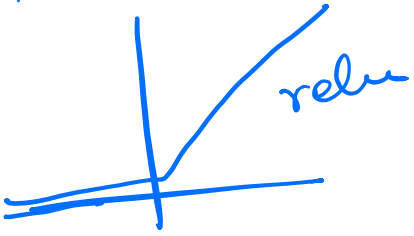
for hidden layers, tanh is better (as mean is closer to 0)

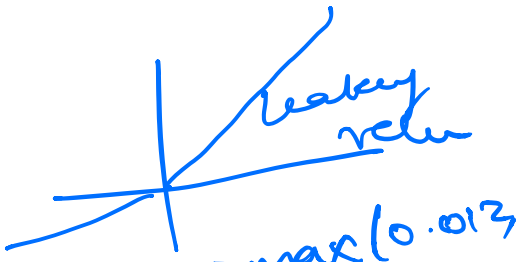
for binary classification, output can use sigmoid as the range is 0 to 1.

large values of z will make learning difficult using tanh, sigmoid.

∴ we can use Relu → (common → rectified linear unit

$f(z) = \max(0, z)$





If activation fn was linear ($g(z) = z$), then $\hat{y}$ becomes linear fn of input features x.

linear activation fns are useful if we are doing regression.. even in that case, hidden layer shouldn't have linear activation (in most cases).

Gradient descent for NN:

Parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$

$n_x = n^{[0]}$ $\quad n^{[1]}, \quad n^{[2]} = 1$

$w^{[1]} \rightarrow (n^{[1]}, n^{[0]})$ $\qquad w^{[2]} = (n^{[2]}, n^{[1]})$

$b^{[1]} \rightarrow (n^{[1]}, 1)$ $\qquad b^{[2]} = (n^{[2]}, 1)$

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}, y)$$

($\hat{y} = a^{[2]}$)

repeat {

compute $\hat{y}^{(i)}$

find $dw^{[1]}, db^{[1]}$

update $w^{[1]}, b^{[1]}$

}

Formulas for computing derivatives

Forward propagation:

$Z^{[1]} = W^{[1]}X + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$

Back propagation:

$dZ^{[2]} = A^{[2]} - Y$ $\quad Y = [y^{(1)}\ y^{(2)} \cdots y^{(m)}]$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$ $\quad (n^{[2]}, 1)$

$db^{[2]} = \frac{1}{m}$ np.sum$(dZ^{[2]}$, axis=1, keepdims=True$)$

$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$ $\quad$ * element-wise product

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

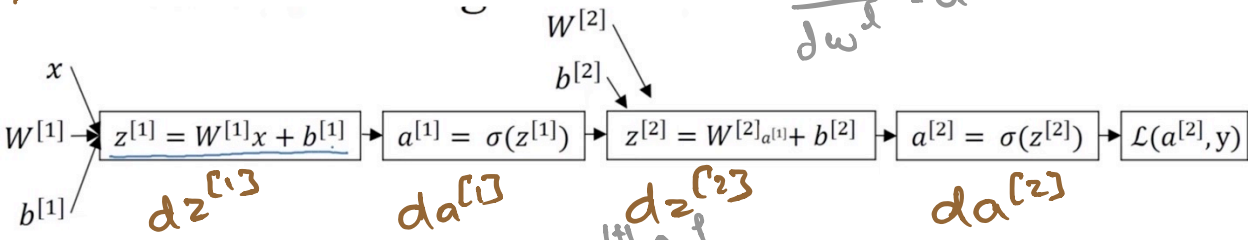
$db^{[1]} = \frac{1}{m}$ np.sum$(dZ^{[1]}$, axis=1, keepdims=True$)$

$(n^{[1]}, 1)$ $\quad (n^{[1]},)$ $\quad$ reshape

Derivation:

$\frac{\partial C}{\partial w^l} = \frac{\partial C}{\partial z^l} \frac{\partial z^l}{\partial w^l}$ $\quad z^l = w^l a^{l-1} + b^l$

$\frac{\partial z^l}{\partial w^l} = a^{l-1}$



$dz^{[1]}$ $\quad da^{[1]}$ $\quad dz^{[2]}$ $\quad da^{[2]}$

$\frac{\partial C}{\partial z^l} = \frac{\partial C}{\partial z^{l+1}} \frac{\partial z^{l+1}}{\partial a^l} \frac{\partial a^l}{\partial z^l}$ $\quad a^{[2]} - y$

$W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$

↗ element wise product

$dw^{[2]} = dz^{[2]} a^{[1]T}$

$db^{[2]} = dz^{[2]}$

$dw^{[1]} = dz^{[1]} x^T$

$db^{[1]} = dz^{[1]}$

$z^{l+1} = w^{l+1} a^l + b^{l+1}$

$\frac{\partial z^{l+1}}{\partial a^l} = w^{l+1}$

$\frac{\partial a^l}{\partial z^l} = g'(z^l)$

Deep NN:

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$

for m training examples:

for $l = 1 \to L$:

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]} \qquad A^{[0]} = X$$

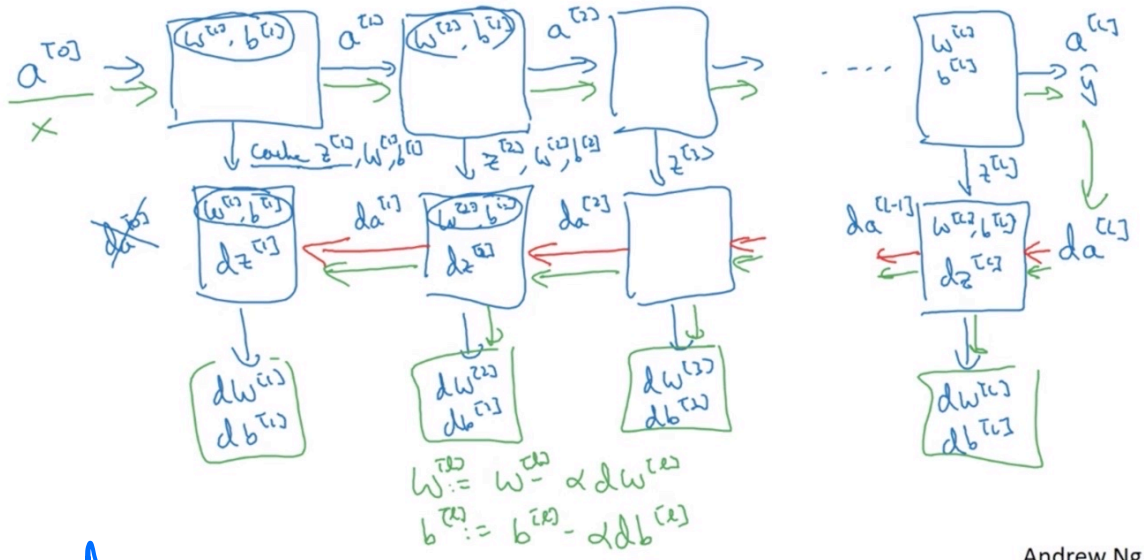
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

$$w^{[l]} \to (n^{[l]}, n^{[l-1]})$$

$n^{[l]}$: no. of neurons in layer l

$$b^{[l]} \to (n^{[l]}, 1)$$

Forward and backward functions



Andrew Ng

backward

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$

$$dw^{[l]} = dz^{[l]} \, a^{[l-1]T}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

$$da^{[L]} = -\frac{y}{a} + \frac{(1-y)}{(1-a)}$$